METHOD AND APPARATUS FOR DOWNSCALING A DIGITAL COLOUR MATRIX IMAGE

The present invention relates to a method and apparatus for downscaling a digital coloured matrix image by selected ratios M_2/M_1 and N_2/N_1, in which the matrix image includes N_1 rows, each row including M_1 pixels, so that the intensity values of the pixels form the matrix and the pixels of different colours form a selected format, and in which scaling is used to form an output matrix, of a size $M_2 \times N_2$, the pixels corresponding to sub-groups of the original matrix, in such a way that $M_2 <= M_1$ and $N_2 <= N_1$, and from the intensity values of which pixels the value of each output pixel of the output matrix is calculated.

Camera sensors are used when using digital cameras to take individual images or a video image. The sensor image can use various image formats, for example RGB8:8:8, RGB5:6:5, YUV4:2:0, and a raw-Bayer image. When the image is displayed in the viewfinder (VF), which usually has a lower resolution than the image sensor, the image must be formed in the sensor and scaled to a suitable resolution for the display. Images can also be zoomed (a smaller image is delimited from the image and then scaled) to the viewfinder. In zooming, there must be many steps from the full image size towards the larger zooming, so that the result of the zooming will appear to be continuous. When video images are coded, the resolution of the video image is also usually lower than the resolution of the sensor. Thus a similar scaling is also required for video. Camera sensors can also be used equally well in portable devices as in cameras.

Image scaling or modifying the size of digital images has been exploited in several digital image-processing applications. The basic method in image scaling is to arrange a two-dimensionally sampled signal in a new sampling array.

A few possibilities for carrying out image sub-sampling are known from the literature on signal and image processing. The sampling of the signal is essential part in the theory of signal processing and has been widely covered in the literature. Basically, it is a matter of preserving the spectrum of the image below Nyquist frequency.

The general sub-sampling methods include an anti-alias filter and the re-creation of the image from the samples. The sampled data is often got using a linear combination of

sampled input data and a specific core.

Sampling algorithms are often compromises between the complexity of the algorithm and the quality of the image obtained.

The simplest form of re-sampling is the 'nearest neighbourhood' method. This does not use anti-alias filtering of the original data, but only selects the nearest samples for the new sampling array. The image obtained as a result is thus poor, due to the anti-aliasing effect.

There are many methods for selecting the coefficients for the core of the anti-alias filter.

US publication 6,205,245 discloses one method, in which a colour image is scaled directly from the sensor's matrix, in such a way that a pixel group, of which one is always processed one at a time, and which corresponds to each pixel of the final image, is defined. In this method, intermediate pixels are jumped over, thus losing original information.

The present invention is intended to create a new high-quality sub-sampling method and an apparatus requiring small amount of memory for implementing the method. The characteristic features of the method according to the invention are stated in the accompanying Claim 1 while the features of the corresponding apparatus are stated in Claim 5. Correspondingly, the features of the software method according to the invention are stated in Claim 14. The use of the method according to the invention requires only small amount of memory and the method is efficient in terms of calculation. The quality of the output image is, however, high. The method is particularly suitable for low-power devices such as cameras, as well as for camera and multimedia communicators. The small memory requirement is due to the fact that, in the scaling, one input image row, i.e. line, which is stored by summing only the amount of a single output line (row) into the line memory, as well as storing in a second line memory that part of the input image line, which is not contained in the output image pixel being processed. In one application, the pair of memories are alternated automatically by addressing the modulo of the index of the output row with the modulo 2-function, i.e. with the least significant bit of the integer (index). In one application, the memory required in scaling is implemented

in the processor (CPU) performing the scaling. This can be a separate DSP (digital signal processor) - circuit component. One example of DSP processor architecture, which can be applied in context with the invention, is the multiprocessor ADSP-BF561 manufactured by the Analog Device Inc. (USA) corporation. A second example referred to here is the RISC processor ARM9EJ-S (product specification: ARM DDI 0222B) of ARM Ltd., UK. In several two-processor applications, the processors are, however, located physically in different circuits/modules.

In the following, the invention is examined with reference to the accompanying figures, which show the method and apparatus of the invention.

Figure 1 shows the scaling concepts
Figure 2 shows an example of an output pixel
Figure 3 shows a depiction of different types of pixels
Figure 4 shows the structure of the memory required in the method
Figure 5a shows the scaling algorithm as a flow diagram
Figure 5b shows the use of DATA memory locations in a most difficult case
Figure 6 shows the raw-Bayer image format and its scaling geometry
Figure 7 shows one apparatus solution as a block diagram
Figure 8 shows the scaling solution of Figure 7 at the circuit level
Figure 9a shows the scaling stages of the raw-Bayer image format
Figure 9b shows, as an alternative output form to that of Figure 9a, combined co-sited RGB,
Figure 10a shows downscaling of the Bayer format to two different formats (co-sited RGB and Bayer),
Figure 10b shows downscaling of the YUV 4:2:0 (H) format,
Figure 10c shows downscaling of the YUV 4:2:2 (H) format, for an individual pixel,
Figure 10d shows downscaling of the YUV 4:4:4 (H) format,
Figure 10e shows downscaling of the YUV 4:4:4 format to a different format YUV 4:2:2.

According to Figure 1, the input image (I_1) and the output matrix (I_2) are placed on top of each other, so that each pixel of the output matrix is influenced by the original pixels. Scaling is a process, in which an output image with a lower resolution is formed from the input image. In order to simplify the depiction, Figures 1 - 5 mainly deal with the

scaling of a black-and-white image. The special features of scaling a coloured image will become apparent later.

The image sizes are I_1: M_1 x N_1 and I_2: M_2 x N_2. The pixels are regarded more as areas than as points. The pixel sizes are set as Sx_1 x Sy_1 for the input image and Sx_2 x Sy_2 for the output image. The image size is defined as L 'units', so that $L = Sx_1M_1 = Sx_2M_2$ = (pixel size) x (number of pixels), so that $Sx_1/Sx_2 = M_2/M_1 = c \times M_2/c \times M_1$.
The pixel sizes are then calculated as follows:
$Sx_1 = c \times M_2$, and
$Sx_2 = c \times M_1$,
c can be chosen freely: for example, c = 1, or $c = 1/gcf(M_1,M_2)$ (gcf: greatest common factor), or $c = 1/M_1$, or any c > 0. It is useful to choose the coefficient c in such a way that Sx_2 is the power of two (1, 2, 4, ...), because divisions by Sx_2 can then be easily performed in binary calculation (i.e. division becomes a transfer in binary integer calculation). The pixel indices are marked as follows: (i = 0, 1, ... , (M_1-1), j = 0, ... , (N_1-1), k = 0, ... , (M_2-1) and l = 0, ... , (N_2-1).

The value of the output pixel is the weighted mean value of the values of the input pixels that 'belong' to the area of the output pixel, Figure 1.

$$Out(k,l) = \left\{ \sum_{(i,j)\in Out(k,l)} W(i,j)In(i.j) \right\} / ScaleFactor$$

The scaling factor ScaleFactor is calculated as follows:

$$ScaleFactor = \sum_{(i,j)\in O\,u\,(k,l)} W(i,j) = \frac{Sx_2}{Sx_1}.\frac{Sy_2}{Sy_1} = \frac{M_1}{M_2}.\frac{N_1}{N_2}$$

W(i.j) is the weighting coefficient. It is relative, depending on how many input pixels there are inside an output pixel:
W(i,j) = the surface area covered by the output pixel (k,l) inside (i, j) / the surface area of the input pixel (i,j).

Figure 2 shows an example of this. The output pixel covers the input pixels: j = 0 ... j_{end}, and i = 0 ... i_{end}. The indices (i=0,j=0) are used to depict the input pixels that initiate an arbitrary output pixel. Some weighting coefficients:
$W(0,0) = (v_{start} \times w_{start}) / (Sy_1 \times Sx_1)$
W(1,1) = 1

$W(1,0) = v_{start}/Sy_1$ ... etc.

The following describes the application of the scaling method.

In order to calculate the output pixel value, it is possible to use separate processing, i.e. the x direction can be calculated first, followed by the y direction:

$Outx(j) = [w_{start}/Sx_1 \times In(0,j) + Sum(In(1,j) \ldots In(i_{end-1},j)) + w_{end}/Sx_1 \times In(i_{end},j)] \times Sx_1 / Sx_2$

$= [w_{start} \times In(0,j) + Sx_1 \times Sum(In(1,j) \ldots In(i_{end-1},j)) + w_{end} \times In(i_{end},j)] / Sx_2.$

$Out = [v_{start}/Sy_1 \times Outx(0) + Sum(Outx(1) \ldots Outx(j_{end-1}) + v_{end}/Sy_1 \times Outx(j_{end})] \times Sy_1 / Sy_2$

$= [v_{start} \times Outx(0) + Sy_1 \times Sum(Outx(1) \ldots Outx(j_{end-1})) + v_{end} \times Outx(j_{end})] / Sy_2.$

This algorithm is based on online processing. One input pixel is processed at a time. The pixel data is summed in the line-memory structure. The weighting coefficients acting on the various indices can be calculated ready in one line memory (preferably the input line memory) in connection with processing, thus further reducing the number of calculation operations. The price for this is, of course, a slight increase in memory, but this is repaid in the form of saved operations. Of course, two line memories will be required, if the vertical and horizontal scaling coefficients are different. When the coefficients are equal, only one (longer) line memory will be required.

The input pixels have a different coefficient, depending on where they are located relative to the output pixel. Thus, there are three types of pixel, see Figure 3.

Full pixel/full pixel row: the input pixel is entirely inside the output pixel.

Final pixel/final pixel row: the input and output pixels have a common end edge.

Part pixel/part pixel row: the area of the input pixel extends to the area of two or four output pixels.

The memory structure is shown in Figure 4. It includes two parts, termed 'Data' and 'Buffer'. The data structure includes four memory locations (Data[0/1][0/1]). The row memory structure includes two line memories (Buffer[0][0 ... M_2-1], Buffer [1][0 ... M_2-1]). The length of the line memories is the width of the output image. Separate memories will always be required for the different colour components when the image is not read many times from one of the large memories and processed only one component at

a time. In the case of a Bayer image, both Gr and Gb are processed as their own colour components and thus require their own memories.

Because the sub-sampling method can be separated into parts, it is always possible to carry out vertical and horizontal scaling in separate processors (for example, in the camera sensor and the base-band processor). One-dimensional scaling does not require a line-memory at all, if the scaling takes place in the direction in which the pixels come into scaling (generally in the horizontal direction). Only a data structure, such as DATA[0/1] is needed, when one-directional sub-sampling should be carried out. This data structure is required on rows or columns including all colour components.

Depending on the type of pixel in question, the DATA and line-memory locations are filled. The relation of the pixel position to the output pixel (k, l) also determines how the memory structure is used (see the pseudocode later).

On the part-pixel line, both line memories Buffer[0][0 ... M_2-1] and Buffer [1][0 ... M_2-1] are in use. This means that the start of the following output line has already been processed. In the part-pixel area, the data structures Data[a][0] and Data[a][1] are summed, in which a can be 0 or 1 (full pixel row), or both 0 and 1 (part pixel row).

Figure 5a shows a flow diagram of the algorithm. We start from the first input row and from the first output row (i,j,k,l=0, stage A). The type of pixel row is derived from the location of the pixel, stage B. The input row is processes according to its type (Stage C, D, or E). After the processing of the row, the next row always comes for processing, until all the rows have been processed. The part pixel row stage E, or the final (full) pixel row on the output row, stage D, terminate the output row. After a full row and a part pixel row, the next output row (l+1) is taken for processing, until the row that was processed was the last one (j= N_1-1), condition G, in which case the scaled image is ready. After this condition, the process either terminates or moves to the following output row, stage F. At the same time, the process moves to the processing of the following input row (j=j+1), stage H, which is reached also from the processing of the full row in the middle of the output row, stage C.

Figure 5b illustrates the operation of the DATA memory locations in a most difficult case, i.e. in the case of a part pixel row. In this figure, the value of the output pixel of column k on row l is processed and in this case the input-image pixels I from 61 – 64 are used in the processing. The even-numbered indices (modulo 2) direct to the first memory location DATA[0][0]. When processing the previous output pixel (k-1), the part intensity (upper broken line) of its last input-image pixel j=61 was stored. Here the 'full' vertical part-intensities (Xup) of the input pixels 62, 63 belonging to the same output pixel k, and this part intensity of the input pixel 64 are summed in it. The final part is stored in the memory location DATA[0][1] to form a 'seed' for the next pixel sum. Simultaneously, in the processing of each pixel i, the Xdown part intensities were summed in the memory location DATA[1][0] and, in the case of the last one, the longitudinal part intensity in the memory location DATA[1][1], to form a seed for the pixel sum of the following row. After this, the contents of the memory locations DATA[0][0] and DATA[1][0] are summed to the elementary units BUFFER[0][k] and BUFFER[1][k] of the row memories. The first of these form the value Output[l][k] and is taken out.

The following described the pseudocode for implementing scaling by software means.

The output row (l) determines basically, which part of the line memory and the Data memory is used (Data[l mod 2] and Buffer[l mod 2], see pseudocode). The number (k) of the output pixel determined the actual memory location (Data[l mod 2] [k mod 2] and Buffer[l mod 2] [k]). In this case, the term 'X mod 2' refers to X modulo 2, i.e. the remainder X when dividing by 2. The memory locations are automatically alternated by taking the least significant bit of the binary integer index as the address.

The following pseudocode depicts how the line-memory structure is exploited.

We use the following notations:

Input pixel: Input[j][i],

Output pixel: Output[l][k],

Weighting coefficients:

$w_{end}(k)$ = FRACTION [(k+1) * M_1/M_2]: FRACTION is a fraction

$v_{end}(l)$ = FRACTION[(l+1) * N_1/N_2]

M_1, M_2: number of pixels in the row

N_1, N_2 number of rows

Sx_1, Sx_2, Sy_1, Sy_2: size of pixels

Scaling factor ScaleFactor (as above)

$w_{start}[k+1] = 1 - w_{end}[k]$ (= $w_{start}/\tilde{Sx_1}$ [0,1]),

$v_{start}[l+1] = 1 - v_{end}[l]$.

For the type of pixel row, the value (= KA) is calculated for the statement ((l+1) x N_1/N_2-j), unless the row in question is the final row (index j = N_1-1). If KA > 1, the row is a full row and in the case KA < 1 it is a part row. If KA = 1, the row is the final row for the relevant output row. The memory recitation commands have been omitted. Each memory location is always reset, when data is forwarded.

Processing of a full pixel row

Full pixel

Data[l mod 2][k mod 2] = Data[l mod 2][k mod 2] + Input[j][i]

Final pixel

Data[l mod 2][k mod 2] = Data[l mod 2][k mod 2] + Input[j][i]

Buffer[l mod 2][k] = Buffer[l mod 2] [k] + Data[l mod 2][k mod 2]

Part pixel

Data[l mod 2][k mod 2] = Data[l mod 2][k mod 2] + Input[j][i] x $w_{end}[k]$

Data[l mod 2][(k+1) mod 2] = Input[j][i] x wstart[k+1]

Buffer[l mod 2][k] = Buffer[l mod 2][k] + Data[l mod 2][k mod 2]

Processing of the final pixel row:

Full pixel

Data[l mod 2][k mod 2] = Data[l mod 2][k mod 2] + Input[j][i]

Final pixel

Data[l mod 2][k mod 2] = Data[l mod 2][k mod 2] + Input[j][i]
Buffer[l mod 2][k] = Buffer[l mod 2]][k] + Data[l mod 2][k mod 2]
Output[l][k] = Buffer[l mod 2][k] / ScaleFactor

Part pixel
Data[l mod 2][k mod 2] = Data[l mod 2][k mod 2] + Input[j][i] x $w_{end}[k]$
Data[l mod 2][(k+1) mod 2] = Input[j][i] x wstart[k+1]
Buffer[l mod 2][k] = Buffer[l mod 2][k] + Data[l mod 2][k mod 2]
Output[l][k] = Buffer[l mod 2][k] / ScaleFactor

Processing of a part pixel row:
Xup = Input[j][i] x $v_{end}[l]$

Xdown = Input[j][i] x $v_{start}[l+1]$

Full pixel
Data[l mod 2][k mod 2] = Data[l mod 2][k mod 2] + Xup
Data[(l+1) mod 2][k mod 2] = Data[(l+1) mod 2][k mod 2] + Xdown

Final pixel
Data[l mod 2][k mod 2] = Data[l mod 2][k mod 2] + Xup
Data[(l+1) mod 2][k mod 2] = Data[(l+1) mod 2][k mod 2] + Xdown
Buffer[l mod 2][k] = Buffer[l mod 2][k] + Data[l mod 2][k mod 2]
Buffer[(l+1) mod 2][k] = Buffer[(l+1) mod 2][k] + Data[(l+1) mod 2][k mod 2]
Output[l][k] = Buffer[l mod 2][k] / ScaleFactor

Part pixel
Data[l mod 2][k mod 2] = Data[l mod 2][k mod 2] + Xup x $w_{end}[k]$
Data[l mod 2][(k+1) mod 2] = Xup x $w_{start}[k+1]$
Data[(l+1) mod 2][k mod 2] = Data[(l+1) mod 2][k mod 2] + Xdown x $w_{end}[k]$
Data[(l+1) mod 2][(k+1) mod 2] = Xdown x $w_{start}[k+1]$
Buffer[l mod 2][k] = Buffer[l mod 2][k] + Data[l mod 2][k mod 2]
Buffer[(l+1) mod 2][k] = Buffer[(l+1) mod 2][k] + Data[(l+1) mod 2][k mod 2]
Output[l][k] = Buffer[l mod 2][k] / ScaleFactor

BAYER IMAGE FORMAT

Scaling can be performed directly from a Bayer format (Figure 6). In a Bayer image, there is only one colour component in one pixel. The sampling frequency of the colour components (R, G_R, G_B, and B) is half of the sampling frequency of the pixels. Thus, the scaling factor for the colour components is twice that of the scaling factor $M_2/M_1 \times N_2/N_1$ between the input and output images, when scaling from a Bayer format to the co-sited RGB format. The scaling factor can be stated in the more general form $f \times M_2/M_1 \times N_2/N_1$, in which f is the factor creating the format conversion. More specifically, the factor $f = f_M \times f_N$, i.e. it too can be divided into horizontal and vertical components. If the format does not change in scaling, then the value of f will be 1 (e.g., scaling from a Bayer format to a Bayer format).

The origin is marked in the figure with a circle in the original format and in the input pixels of each colour. If, due to the offsets of the input image, pixel data does not exist for the entire area, then in that case it is advantageous to extrapolate slightly more the value of the previous pixel, so that there will be no need to alter the total scaling ratio in different ways for the edge pixels. Examples of the scaling of different formats are given below.

The offset of the output image can be implemented using simple counters or adjusting the different colour components to be the same. This means that the output image is offset relative to the input image. In practice, this means that part of the information of the edge pixels of the input image are left unused while correspondingly additional information must be extrapolated at the opposite edge.

The invention is particularly suitable for hardware-based applications, one example of which is the application according to Figure 7. The camera module 10 is connected to a host system 22, which controls the display device 24 and the camera module. The camera module 10 includes particularly optics, i.e. a lens arrangement 11 (in practice several lenses), a sensor 12, an image-processing circuit 14, a scaling unit 16, and a controller component 20. The image-processing circuit 14 reads, in a known manner, the sensor 12, so that a high-speed data stream is created, which is led to the scaling component 16, from which the data stream of the scaled image is led to the host system 22.

In this case, the memory requirement is about two input-image lines. This means, for example, Cx2x160 words in a 160x120 matrix, in which C is the number of colour components (generally 3 - for RGB or YUV images). The length of the data word depends on the precision of the calculation and is, for example, 2 or 4 bytes.

At the circuit level, the structure of the scaler is in one (fully digital) application according to Figure 8. The scaler 16 includes an input component 161, a CPU 162, a memory 163, an output component 164, and a control component 167, all connected to an internal bus 165. This output component 164 feeds the scaled output pixel values (Output[l][k] to the host system 22.

The apparatus can include a scaler component, in which there are separate processors (CPUs) for scaling in each dimension (for example, the aforementioned ADSP-BF561). The memory can be memory that is physically inside the processor.

In one application (scaling ratio less than ½), the apparatus includes memory for the scaling function for at most one image sensor line for each colour component.

The apparatus is highly suitable for a mobile terminal.

The following are various alternative implementations of the invention. If there are two devices in the system (a camera module and the host system), the scaler can be implemented in very many different places and in many different ways.

1. The sensor image is processed and scaled by the module and is then sent to the host system (Figure 7).
2. The sensor image is scaled and processed by the module and then sent to the host system.
3. The sensor image is processed by the module and is then sent to the host system, in which it is scaled.
4. The sensor image is scaled by the module and then it is sent to the host system, in which it is processed.
5. The sensor image is sent to the host system, in which it is scaled and processed.
6. The sensor image is sent to the host system, in which it is processed and scaled.

Scaling can always also be carried out in the display device, provided it has a processing capacity. In addition, the image need not come directly from the sensor/camera module, but instead it can also be read from the memory/memory card. The image can also be compressed (e.g, JPEG), in which case encoding is performed first, followed by scaling. These stages too (decoding and scaling) can be combined. In addition, situations can also be found, in which the scaling can be carried out in two parts, in such a way that first of all the image is scaled in the horizontal direction by the camera module and then sent to the host system, in which it is scaled in the vertical direction and then processed. In addition, the scaler can be used in several (2) different stages, so that this small-memory scaler replaces the previous implementation of the scaler. In addition, the scaler can be used in such a way that a small image is rapidly formed on the display (using simple processing), and only then is the actual larger output image processed. If scaling takes place in the camera module, the host system must of course tell what size of image is required on the display. In addition, scaling may also be required in situations, in which a video image is stored in the memory through a video encoder. In addition, during storing it may be possible to view a viewfinder image on a differently sized display. Thus, situations can be found, in which the video encoder and not the display determines the output image. Similarly, there may be a need to use the same kind of scaler at two different ratios, either sequentially or in parallel, for different purposes.

According to Figure 9a, the colour components Gr and Gb, like R and B, are processed as totally separate components, which can then be finally combined. According to Figure 9a, initially four separate components are processed, which form the elementary units of the scaled image directly in the Bayer format. When the output information is co-sited R, G, and B (Figure 9b), the two G components are combined into a single G component. (It is not essential to combine them when scaling to co-sited, but this is often worth doing, in order to minimize the amount of data to be sent.) In order to keep the overall logic of the scaler simple, the G components are processed separately. Thus the component order is always retained in scaling (= the order of the incoming data is the same as the order of the outgoing data). It is important only to know that the question is of a Bayer as an input image. It should also be known whether the output image is Bayer or co-sited, so that the correct offsets (and scaling factors) will be used. This concerns other colour formats as well, such as YUV and RGB. The order of the output components will only change if a switch is made to a different sub-sampling (e.g., from

YUV4:2:2 to YUV4:4:4), or a change, e.g., from an interleaved format to a progressive format takes place while scaling .

Because scaling can perfectly well be performed separately (first in the x direction and then in the y direction), and because scaling can only be performed in one direction, it is sometimes clearer and simpler to consider the scaling factors only in one direction.

In certain image formats (YUVx:x:x, Bayer RGB, co-sited RGB, etc., interleaved/planar formats) the pixel data is read from a data flow, a memory, or somewhere else, in a predefined order, or correspondingly the pixel data is written to a data flow, memory, or somewhere else in a predefined order. In Figures 10a – 10e, this data flow is marked 'INPUT data order' or 'OUTPUT data order'. In the data order, the size of the pixel is of no importance, but for reasons of clarity the pixels are drawn in the figures according to the scaling ratio.

In scaling according to the patent application, it is, however, important to know the positions and areas of the pixels (Figure 1 and Figure 2). The generally known image formats also define the positions and areas of the pixels and these are marked 'INPUT pixel positions' or 'OUTPUT pixel positions' in Figures 10a – 10e. The scaling geometry is intended to match the input and output pixels with each other (according to Figures 1 and 2).

The broken line on the left-hand side in the figures refers to the origin, point (0, 0). Generally, the term origin refers to the left-hand upper corner of the image. In figures 10a – 10e, a few of the first pixels are shown, starting from the origin.

In the following images $S_1 = Sx_1$ or $S_1 = Sy_1$, and $S_2 = Sx_2$ or Sy_2 (as in Fig. 2) according to the situation. S_1 and S_2 refer to the size of the pixels.

Figure 10a shows the same Bayer-image scaling geometry as Figure 6. Due to the construction of a Bayer-image sensor, the colour components have their own well-defined position. Due to this, the colour components are moved suitably relative to the origin (in the image by the amount $S_1/2$, $S_2/2$, Sx/2, or Sy/2), so that the information of the colour components will also match each other. In this figure, the question is of Bayer to co-

sited RGB and Bayer-to-Bayer scaling. In this case, it should be noted that when an image is scaled to the Bayer format, the image obtained after scaling can be interpolated, by CFA interpolation, to the size indicated by the number of pixels. When scaling to co-sited format, CFA interpolation is not required, by size of the output image is halved from the number of pixels after scaling by finally combining the components. If the resolution of the initial image is thus, e.g., 1600 x 1200 pixels and the scaling factor S_1/S_2 is 0.5 (= ½) in both directions, then the size of the output image of Bayer scaling will be 800 x 600 pixels, from which an output image of a size 800 x 600 can be interpolated. Correspondingly, when scaling to co-sited format, the size of the scaling output image will also be 800 x 600, from which a (4x)400 x 300-sized output image can be created. Correspondingly, if, in the co-sited format, the G-components are combined, then the size of the output image after scaling would be (3x)400 x 300 (i.e. 25 % fewer pixels than without combining). This leads to the doubling of the scaling factor referred to earlier. Thus the real scaling factor S_2/S_1 is actually $2 \times (S_2/S_1)$, that is, if this is compared using the ratios of the sizes of the output images created (i.e. for example, the scaling factor for a co-sited image is in practice, in both the x and the y direction 1 / scalefactor = 0.25 = 400 / 1600 = 300 / 1200). Thus, this does not affect the scaling factor and calculations during the actual scaling, but only the ratios of the output images.

Figure 10b – c, e. In the formats YUV4:2:0 and YUV4:2:2, the sampling frequency of the U and V components is half the sampling frequency of the Y component, in which case the size of a U or V pixel will be two times ($2S_1$) the size (S_1) of a Y pixel. In the YUV4:1:1 format, the sampling frequency of a U or V component is one-quarter of the sampling frequency of a Y component, so that correspondingly the size of a U or V pixel will be fours time the size of a Y pixel.

Figure 10b. In the YUV4:2:0 format, only either the U or the V component is stored on each line. Thus a YU line or YV line can be processed in the same way. The notation U/V means that the pixel can be either a U pixel or a V pixel per line.

Figure 10c shows the processing of an individual pixel somewhere in the middle of the scaling of the image YUV 4:2:2 (H). The scaling YUV 4:2:2 -> YUV 4:2:2 vertical is the same as YUV 4:4:4 -> 4:2:2.
In this case, the values of the output pixels are calculated as follows:

$Yout = [W_{START.Y} \cdot Yin(1) + S_1 \cdot Yin(2) + W_{END.Y} \cdot Yin(3)]/S_2$

$Vout = [W_{START.V} \cdot Vin(1) + 2S_1 \cdot Vin(2) + W_{END.V} \cdot Vin(3)]/2S_2$ (and U as for V)

Figure 10d shows YUV 4:4:4 -> YUV 4:4:4 scaling. (Co-sited RGB scaling is the same as YUV 4:4:4, if R, G, and B are written in place of Y, U, and V.)

Figure 10e shows YUV 4:4:4 -> YUV 4:2:2 scaling.

A simultaneous sub-sampling variation and suitable scaling can be easily derived from the above examples. An example of this would be YUV 4:2:0 to YUV 4:4:4 conversion, in which scaling takes place by 0.5 in both the X and the Y direction. In that case, only the Y component need be scaled while the colour components remain unchanged.

By using this method, the same advantages as in other image scaling can be obtained in scaling a Bayer image. This are the correct use of the image information and the best possible filtering. In addition, Bayer scaling will substantially reduce the total calculation requirement and data traffic between processing. The method described to scale all components separately also brings the advantage that the precise order of the colour components need not be known in the scaler and thus, for example, image rotations, mirrorings, or sensor readings in different directions can be taken care of easily, without having to alter the scaler or its operation. In addition, when using different offsets and component settings, the scaler can be used for scaling very different kinds of colour-image information, and also for creating different kinds of output format. It can also be used to generate, for example, either Bayer or co-sited output images as required.